This is the author's version of this paper and psoted here for personal use only. For the final version please refer to the ACM Digital Library:
https://doi.org/10.1145/3183440.3190328

When to Extract Features: Towards a Recommender System

Jacob Krüger
Otto-von-Guericke University Magdeburg & Harz University of Applied Sciences Wernigerode, Germany
jkrueger@ovgu.de

ABSTRACT

In practice, many organizations rely on cloning to implement customer-specific variants of a system. While this approach can have several disadvantages, organizations fear to extract reusable features later on, due to the corresponding efforts and risks. A particularly challenging and poorly supported task is to decide which features to extract. To tackle this problem, we aim to develop a recommender system that proposes suitable features based on automated analyses of the cloned legacy systems. In this paper, we sketch this recommender and its empirically derived metrics, which comprise cohesion, impact, and costs of features as well as the users' previous decisions. Overall, we will facilitate the adoption of systematic reuse based on an integrated platform.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; **Software reverse engineering**; *Risk management*;

KEYWORDS

Software product line, extractive approach, software maintenance

ACM Reference Format:
Jacob Krüger. 2018. When to Extract Features: Towards a Recommender System. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3183440.3190328

1 INTRODUCTION

Software reuse is one of the most important concepts in software engineering, reducing development and, if done systematically, maintenance costs [3, 30]. However, organizations mostly apply reuse by cloning and then adapting systems to new customer requirements [3, 8, 28]. This is referred to as *clone-and-own* approach, which is easy to use and requires no initial planning. Still, this approach also results in several separated clones that must be maintained. Here, the problem of change propagation [24] to introduce new features or to fix bugs can drastically increase the maintenance costs, as each variant must be considered individually.

A systematic approach to reuse variants are software product lines (SPLs) [3, 6]. In an SPL, features (e.g., movement commands [4, 15]) are used to describe common and variable functionalities of variants. These features are modeled in a variability model [7] – to document their interdependencies – and are implemented only once. Instead of cloning and changing all code for each new variant, only new features are introduced into the SPL. Then, a valid set of features, a configuration, is selected to derive a variant. In practice, SPLs are mostly implemented with preprocessors that annotate features in a single code base [3, 21]. However, preprocessors usually do not allow to physically separate and compose features.

Physical separation promises several benefits, such as, improved traceability and modularity [3]. Despite such promises, organizations fear to migrate from cloned variants towards a composition-based SPL, due to the corresponding costs and risks [14, 28]. **A particular challenge is to decide which features are suitable and necessary to extract from the legacy systems.** Numerous factors influence such a decision including, for example, a feature's granularity, its usage among variants, its extraction costs, and the potential savings. Consequently, these decisions are heavily based on intuition, as empirical data and tools are still missing [13, 29].

To tackle this problem, we conduct empirical studies on physical separation of features. We identify characteristics and thresholds to support the decision whether a feature is suitable to extract from a legacy system. In this paper, we describe our overall goal: We aim to implement a recommender that analyzes characteristics of the features in legacy systems and assesses their potential for extraction. The metrics and thresholds can be customized by developers, but we aim to provide default values. Thus, we facilitate the extraction of features into a reusable, composition-based platform.

2 THE RECOMMENDER

With our recommender, we aim to analyze features in legacy systems and suggest those that promise benefits if they are physically separated. In contrast to solely intuitive decisions, we will base our recommender on empirical data.

Input For the first version, we solely rely on the source code of the legacy systems: All features of interest have to be annotated in at least one system. Feature location [26] and code clone detection [25] can be combined to locate and mark the code of optional and mandatory features [11, 12, 15, 17]. These annotations allow our recommender to identify all features and compute metrics for them. In later versions, we will include additional information sources as input, for example, version control systems. Thus, we aim to include previous decisions of the developer and collaborators.

Metrics Deciding whether feature extraction may be beneficial is a challenging task including several uncertainties and metrics. Consequently, we can hardly include all metrics, but will support developers to add additional ones, if necessary. Based on our research [14–19], we will initially include the following metrics:

Size, scattering, and tangling: These metrics are well established to measure the size and distribution of a feature in the source code. Our findings [15, 16, 18, 19] indicate that these metrics are essential

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3190328

to decide when to extract a feature. For example, our studies show that larger, cohesive units of code that have few dependencies with the remaining system are preferable to extract. These metrics also seem to differ for optional and mandatory features, potentially allowing our recommender to identify this property automatically.

Impact analysis: It is important to identify how a feature is integrated within the SPL. Features that are used by several or all (i.e., mandatory) variants are more useful to extract than those that are specific for a single variant [15, 17]. In addition, dependencies to other features indicate how many changes are necessary to extract a feature and still allow to instantiate the legacy systems. Later on, providing an additional feature model can extend this analysis.

Costs and savings: For organizations, costs and savings are the most important metrics [5]. Only if extracting a feature promises a reasonable relation of risks, investments, and benefits, an organization can be convinced. Despite research on this topic, empirical data on the costs of feature extraction are still missing [14, 18]. To gain such data, we conduct empirical studies in organizations. Here, we analyze SPL engineering, clone-and-own approaches, and the migration to abstract associated tasks and costs of each scenario. Based on these studies, we aim to derive a model that translates our other metrics into costs by considering the necessary tasks.

For each metric, we aim to provide computations and thresholds based on empirical studies, but also allow users to adapt these to their needs. Later on, we will include further metrics and additional information sources, for example, based on version control systems.

Output As output for our recommender, we envision information sheets that summarize the metrics for each feature [15]. These sheets shall contain estimations for the costs of extraction and potential benefits. A ranked list or matrix of features – using normalized metrics – can be generated to provide an overview of all features. Here, the developer can mark (e.g., suitable) features and the recommender shall use such markings to rank other features.

3 DISCUSSION

Novelty We are unaware of any recommender to automatically analyze legacy systems to decide which features are suitable for physical separation. Our goal is to provide such a recommender that can be included into any analysis tool and process. While some of our metrics are well-known in the SPL community, most still need empirical investigations, for example, considering mandatory features, their dependencies, and costs [14, 15]. Consequently, feature extraction is often solely based on intuition instead of empirical data. Our research provides insights into the usefulness of specific metrics, the importance of features in an SPL's architecture, as well as extraction costs. Overall, we facilitate the introduction of systematic reuse based on composition into practice.

Contributions Overall, our contributions and results will include:

- Empirical analyses on features and their characteristics. Our analyses include, for example, code comprehension, architectural alignment, and costs [14, 15, 19]. The results help to better understand such factors, supporting practitioners and further research on feature-oriented software reuse.
- A set of metrics and thresholds derived from empirical studies [13, 15, 16, 18]. We aim to utilize experiments but also developers' experiences to derive recommended thresholds.
- A model to describe tasks and costs of extracting features from legacy systems [14]. Thus, we help organizations to better understand corresponding costs, risks, and benefits.
- A recommender system that includes these contributions in a single tool. We will implement it in a way that facilitates reuse and extensibility for different projects and processes.

Overall, our contributions help to better understand composition-based software reuse. We facilitate the adoption of such approaches and support organizations in making a reasonable decision.

Ongoing Work Our main task for future research is to conduct further studies on different characteristics of software features and their extraction. Currently, we are focusing on practical investigations on tasks, costs, and experiences of organizations that have extracted features. Here, we aim to refine our metrics and identify suitable thresholds. Based on the results, we will implement our recommender system and potentially integrate it into an existing tool to facilitate evaluations. Consequently, parts of our research also investigate the initial steps of feature location and following steps of the actual extraction.

4 RELATED WORK

We are unaware of a recommender that is based on empirical data and supports the decision whether to extract a feature. Some recommender systems for SPLs investigate the extraction of variability models from legacy artifacts [10, 27], support configuration processes [23], or propose variability points in general [31]. Closest to our approach seems to be VarMeR [31], which focuses on visualizing commonalities and variability between cloned variants and proposes polymorphism to improve reuse. In contrast, we focus on a feature-oriented notion instead of sole code reuse, base our recommendations on empirical data, and incorporate cost estimations.

Some approaches aim to automatically detect refactoring opportunities in source code [1, 9]. Mostly, the purpose of such approaches is to improve the source code and to remove code smells or code clones. Still, the defined metrics and detection approaches can complement our work on physical separation of features.

There exist empirical studies on features and their characteristics [20, 21, 29] as well as on costs of SPL adoption [2, 22]. However, the existing studies are often focused solely on variable features and also limited in their validity [13, 15, 29]. Similarly, current cost models have shortcomings considering empirical data, practical evaluations, and focus on SPL extraction [2, 14]. We base our research on such works, but will complement and extend them.

5 CONCLUSION

Physically separating features from legacy systems promises benefits, but is also connected to costs and risks. In practice, features are typically separated based on intuition, as metrics and data to reason about this approach are missing. Within this paper, we sketched a recommender system to facilitate such decisions. Based on automated analyses and developers' preferences, it ranks features that are suitable for extraction and estimates corresponding efforts. To provide a substantial basis, we conduct several empirical studies to scope our recommender and provide a practically useful tool.

Acknowledgments Supported by DFG grant LE 3382/2-1. I thank Thomas Leich, Gunter Saake, Thorsten Berger, and Regina Hebig.

REFERENCES

[1] Jehad Al Dallal. 2015. Identifying Refactoring Opportunities in Object-Oriented Code: A Systematic Literature Review. *Information and Software Technology* 58 (2015), 231–249.

[2] Muhammad Sarmad Ali, Muhammad Ali Babar, and Klaus Schmid. 2009. A Comparative Survey of Economic Models for Software Product Lines. In *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 275–278.

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines.* Springer.

[4] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. 2015. What is a Feature? A Qualitative Study of Features in Industrial Software Product Lines. In *International Conference on Software Product Line*. ACM, 16–25.

[5] Barry W. Boehm. 2002. Software Engineering Economics. In *Software Pioneers: Contributions to Software Engineering*. Springer, 641–686.

[6] Paul Clements and Linda Northrop. 2002. *Software Product Lines.* Addison-Wesley.

[7] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *International Workshop on Variability Modelling of Software-Intensive Systems.* ACM, 173–182.

[8] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *European Conference on Software Maintenance and Reengineering*. IEEE, 25–34.

[9] Wolfram Fenske, Jens Meinicke, Sandro Schulze, Steffen Schulze, and Gunter Saake. 2017. Variant-Preserving Refactorings for Migrating Cloned Products to a Product Line. In *International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 316–326.

[10] Mostafa Hamza and Robert J. Walker. 2015. Recommending Features and Feature Relationships from Requirements Documents for Software Product Lines. In *International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. IEEE, 25–31.

[11] Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. 2015. Maintaining Feature Traceability with Embedded Annotations. In *International Systems and Software Product Line Conference*. ACM, 61–70.

[12] Sebastian Krieter, Jacob Krüger, and Thomas Leich. 2018. Don't Worry About it: Managing Varaibility On-The-Fly. In *International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, 19–26.

[13] Jacob Krüger. 2017. Lost in Source Code: Physically Separating Features in Legacy Systems. In *International Conference on Software Engineering Companion*. IEEE, 461–462.

[14] Jacob Krüger, Wolfram Fenske, Jens Meinicke, Thomas Leich, and Gunter Saake. 2016. Extracting Software Product Lines: A Cost Estimation Perspective. In *International Systems and Software Product Line Conference*. ACM, 354–361.

[15] Jacob Krüger, Wanzi Gu, Hui Shen, Mukelabai Mukelabai, Regina Hebig, and Thorsten Berger. 2018. Towards a Better Understanding of Software Features and Their Characteristics: A Case Study of Marlin. In *International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, 105–112.

[16] Jacob Krüger, Kai Ludwig, Bernhard Zimmermann, and Thomas Leich. 2018. Physical Separation of Features: A Survey with CPP Developers. In *Symposium on Applied Computing*. ACM, 2028–2035.

[17] Jacob Krüger, Louis Nell, Wolfram Fenske, Gunter Saake, and Thomas Leich. 2017. Finding Lost Features in Cloned Systems. In *International Systems and Software Product Line Conference*. ACM, 65–72.

[18] Jacob Krüger, Marcus Pinnecke, Andy Kenner, Christopher Kruczek, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2018. Composing Annotations Without Regret? Practical Experiences Using FeatureC. *Software: Practice and Experience* 48, 3 (2018), 402–427.

[19] Jacob Krüger, Ivonne Schröter, Andy Kenner, Christopher Kruczek, and Thomas Leich. 2016. FeatureCoPP: Compositional Annotations. In *International Workshop on Feature-Oriented Software Development*. ACM, 74–84.

[20] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. 2010. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In *International Conference on Software Engineering*. ACM, 105–114.

[21] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Sarah Nadi, and Rohit Gheyi. 2015. The Love/Hate Relationship with the C Preprocessor: An Interview Study. In *European Conference on Object-Oriented Programming*, Vol. 37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 495–518.

[22] Andy J. Nolan and Silvia Abrahão. 2010. Dealing with Cost Estimation in Software Product Lines: Experiences and Future Directions. In *International Systems and Software Product Line Conference*. Springer, 121–135.

[23] Juliana Alves Pereira, Pawel Matuszyk, Sebastian Krieter, Myra Spiliopoulou, and Gunter Saake. 2016. A Feature-Based Personalized Recommender System for Product-line Configuration. *SIGPLAN Notices* 52, 3 (2016), 120–131.

[24] Tristan Pfofe, Thomas Thüm, Sandro Schulze, Wolfram Fenske, and Ina Schaefer. 2016. Synchronizing Software Variants with VariantSync. In *International Systems and Software Product Line Conference*. ACM, 329–332.

[25] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Science of Computer Programming* 74, 7 (2009), 470–495.

[26] Julia Rubin and Marsha Chechik. 2013. A Survey of Feature Location Techniques. Springer, 29–58.

[27] Abdel Salam Sayyad, Hany Ammar, and Tim Menzies. 2012. Software Feature Model Recommendations Using Data Mining. In *International Workshop on Recommendation Systems for Software Engineering*. IEEE, 47–51.

[28] Klaus Schmid and Martin Verlage. 2002. The Economic Impact of Product Line Adoption and Evolution. *IEEE Software* 19, 4 (2002), 50–57.

[29] Janet Siegmund, Christian Kästner, Jörg Liebig, and Sven Apel. 2012. Comparing Program Comprehension of Physically and Virtually Separated Concerns. In *International Workshop on Feature-Oriented Software Development*. ACM, 17–24.

[30] Thomas A. Standish. 1984. An Essay on Software Reuse. *IEEE Transactions on Software Engineering* SE-10, 5 (1984), 494–497.

[31] Anna Zamansky and Iris Reinhartz-Berger. 2017. Visualizing Code Variabilities for Supporting Reuse Decisions. In *Symposium on Conceptual Modelling Education*. 25–34.